Homework 2

学号：16340294

姓名：张星

【Exercise 1】

由于本题有两个自变量，一个因变量，所以我采用三个参数来训练回归模型，具体公式为：

$$\mathrm{z} = \mathrm{ax} + \mathrm{by} + \mathrm{c}$$

其中 a, b, c 为我们待训练的三个参数，x, y, z 三个变量分别表示面积，距离以及价格。每次迭代的公式为：

$$\mathrm{a} = \mathrm{a} - \mathrm{learningRate}\frac{1}{m}\sum_{i=0}^{m}(h_\theta(x^{(i)}, y^{(i)}) - z^{(i)})x^{(i)}$$

$$\mathrm{b} = \mathrm{b} - \mathrm{learningRate}\frac{1}{m}\sum_{i=0}^{m}(h_\theta(x^{(i)}, y^{(i)}) - z^{(i)})y^{(i)}$$

$$\mathrm{c} = \mathrm{c} - \mathrm{learningRate}\frac{1}{m}\sum_{i=0}^{m}(h_\theta(x^{(i)}, y^{(i)}) - z^{(i)})$$

训练样本与测试样本的误差：

$$\mathrm{cost} = \frac{1}{m}\sum_{i=0}^{m}(h_\theta(x^{(i)}, y^{(i)}) - z^{(i)})^2$$

归一化公式：

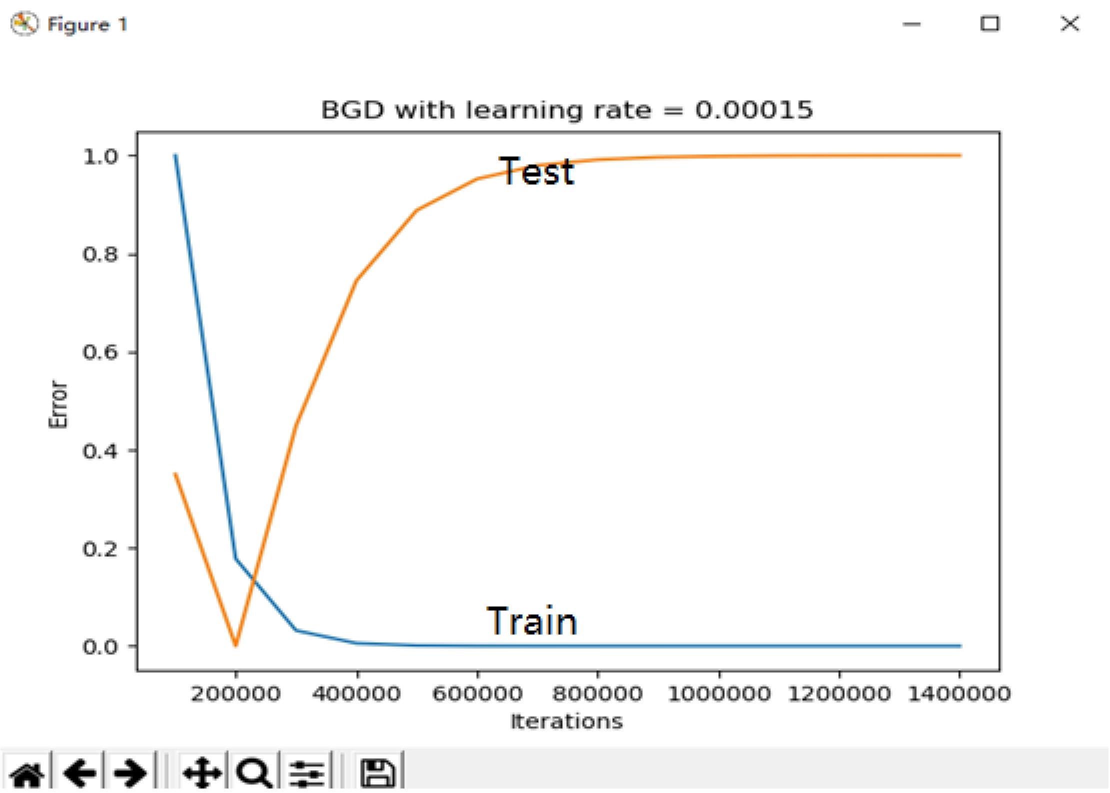
$$\mathrm{z} = \frac{x - min}{max - min}$$

标准化公式：

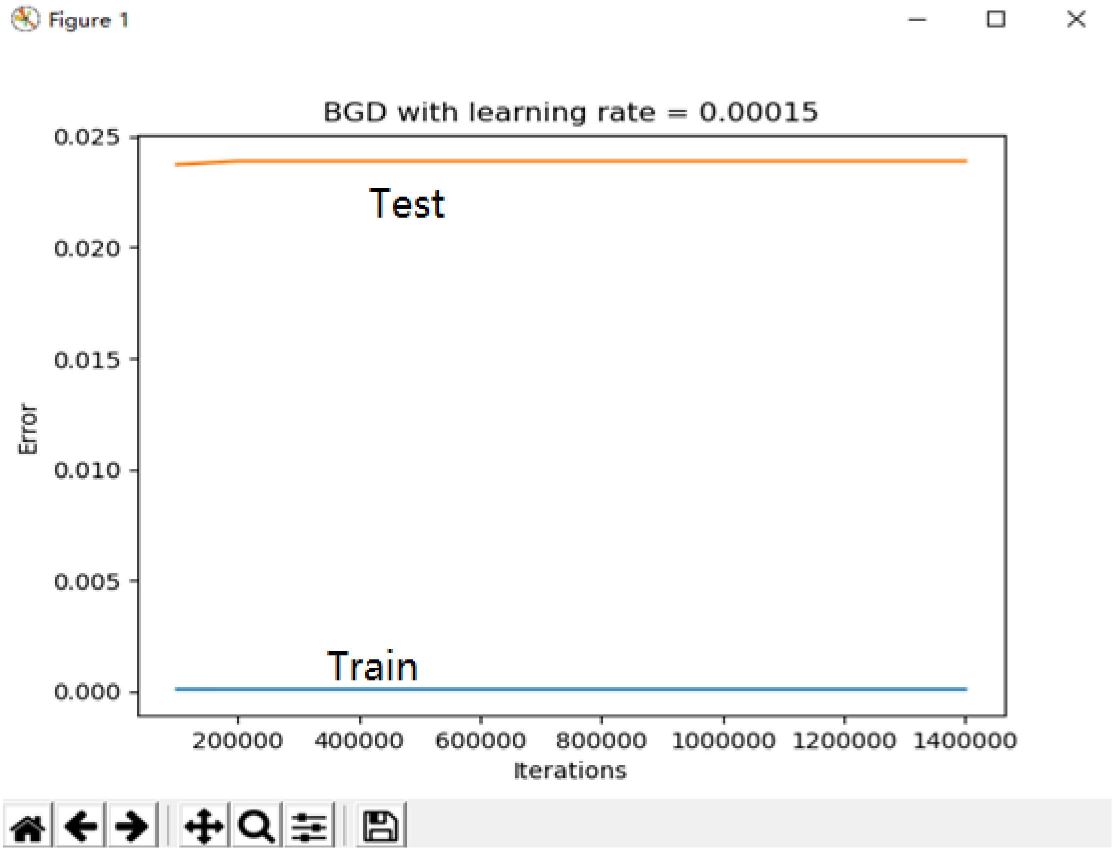
$$\mathrm{z} = \frac{x - \mu}{\sigma}$$

首先，我没有对训练集和测试集数据进行标准化，训练结果如下，误差进行了归一化：由图可知，对于训练集的测试结果还算不错，但是对于测试集来说训练结果就很差。我们的目的是尽可能提高泛化能力，所以这样的结果不符合我们的期望。

对于未标准化的数据，训练集的损失不断下降，说明梯度下降法在不断逼近局部最优解，直至最后稳定，说明维持在最优解附近。测试集则是一开始不断逼近，但是到后面不断远离最优解，这是因为学习率太大的缘故，导致无法收敛，越来越偏离最优解。



接着我又对训练集数据和测试集数据进行了标准化，因为训练的三个参数相差太大，导致收敛速度不一致，并且所占权重也各有不同，所以需要进行标准化。训练的结果很好，一开始就十分稳定：标准化的数据训练结果很好，很快就达到了收敛，对于测试集也是如此，所以标准化能够加速收敛过程。



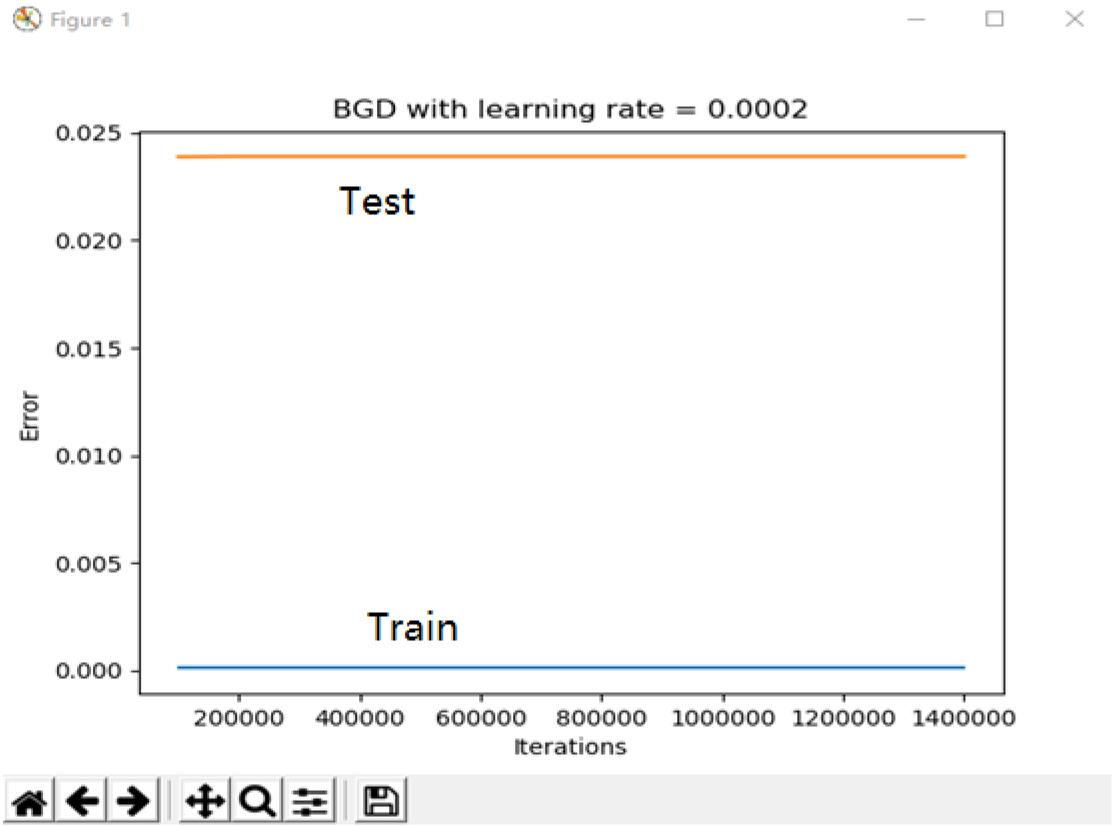
代码：

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import *


# z = ax+by+c

def standardize(X):
    m, n = X.shape
    for i in range(0, n):
        arr = X[:, i]
        meanVal = arr.mean(axis=0)
        std = arr.std(axis=0)
        if std != 0:
            X[:, i] = (arr-meanVal)/std
        else:
            X[:, i] = 0
    return X

def test(a, b, c, data):
    length, n = data.shape
    error = 0.0
    for i in range(0, length):
        x = data[i, 0]
        y = data[i, 1]
        z = data[i, 2]
        f = (a * x) + (b * y) + c
        error += (1/float(length))*np.power((f - z), 2)
    return error

def gradient_descent(a, b, c, data, learning_rate):
    gradient_a = 0.0
    gradient_b = 0.0
    gradient_c = 0.0
    length, n = data.shape

    for i in range(0, length):
        x = data[i, 0]
        y = data[i, 1]
        z = data[i, 2]
        gradient_a += (1/float(length)) * (((a * x) + (b * y) + c) -
z) * x
```

```
        gradient_b += (1/float(length)) * (((a * x) + (b * y) + c) - 
z) * y
        gradient_c += (1/float(length)) * (((a * x) + (b * y) + c) - 
z)
    next_a = a - (learning_rate * gradient_a)
    next_b = b - (learning_rate * gradient_b)
    next_c = c - (learning_rate * gradient_c)
    return [next_a, next_b, next_c]
def run():
    trainingData = np.genfromtxt('dataForTraining.txt',
                     delimiter=' ')
    testData = np.genfromtxt('dataForTesting.txt',
                           delimiter = ' ')
    learning_rate = 0.00015

    trainingData = standardize(trainingData)
    testData = standardize(testData)
    a = 0.0
    b = 0.0
    c = 0.0
    num_iteration = 1500000
    errorForTraining = []
    errorForTesting = []
    for i in range(0, num_iteration):
        a, b, c = gradient_descent(a, b, c, trainingData, 
learning_rate)
        if i % 100000 == 0 and i != 0:
            errorForTraining.append(test(a, b, c, trainingData))
            errorForTesting.append(test(a, b, c, testData))

    X = []
    for i in range(1, 15):
        X.append(i*100000)
    plt.figure(1)
    plt.plot(X, errorForTraining)
    plt.plot(X, errorForTesting)
    plt.title('BGD with learning rate = 0.00015')
    plt.xlabel('Iterations')
    plt.ylabel('Error')
    plt.show()
if __name__ == '__main__':
    run()
```

【Exercise 2】

本题与上次的唯一不同就是学习率有别。由于学习率的提升，导致如果数据不先进行标准化的话，就会出现溢出的情况。所以我直接将其标准化，然后看收敛速度：有图可见，虽然最终的结果与第一题相差不大，但还是能够看出来针对测试集比第一题收敛稍快：与上一题相同，标准化后的数据训练结果很好，测试集也比上次更早收敛，因为学习率比上次大的缘故。但是有些数据若不进行标准化，是无法收敛的，导致训练时出现数据溢出的情况。



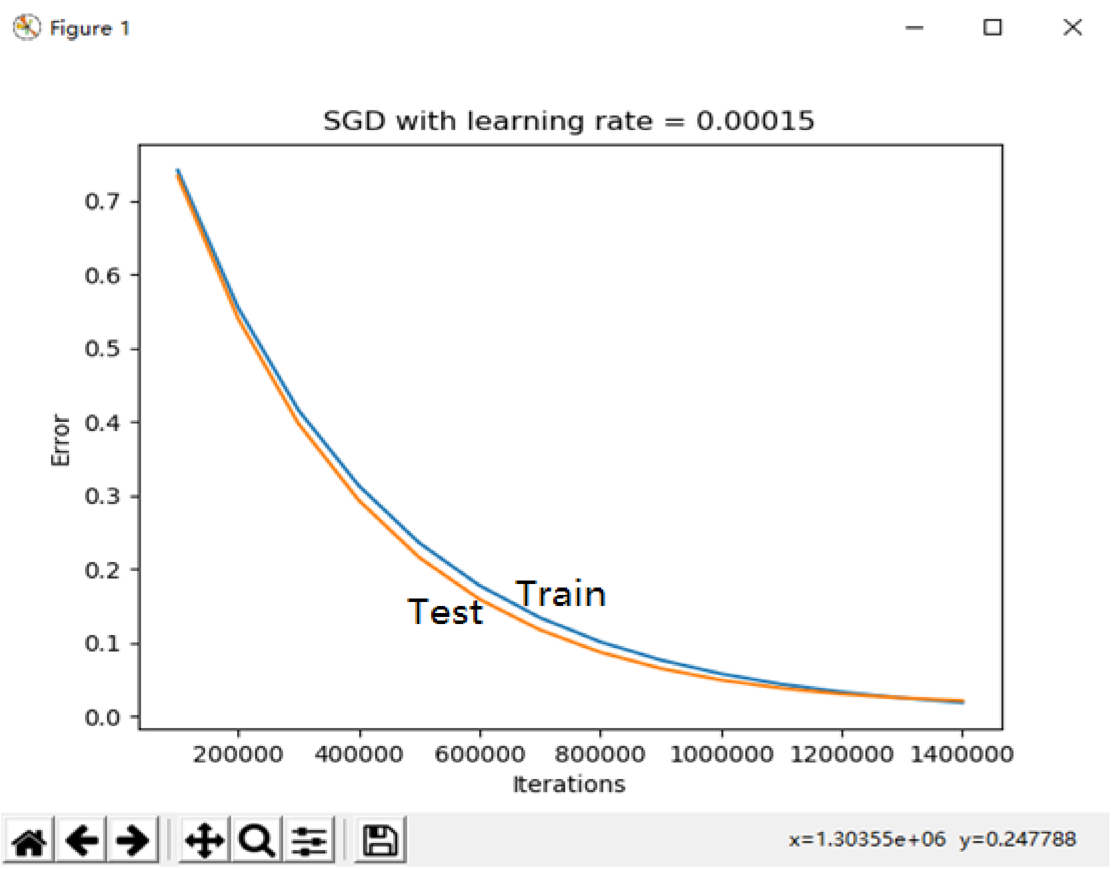
代码：

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import *
# z = ax+by+c

def standardize(X):
    m, n = X.shape
    for i in range(0, n):
        arr = X[:, i]
        meanVal = arr.mean(axis=0)
        std = arr.std(axis=0)
        if std != 0:
            X[:, i] = (arr-meanVal)/std
        else:
            X[:, i] = 0
    return X
```

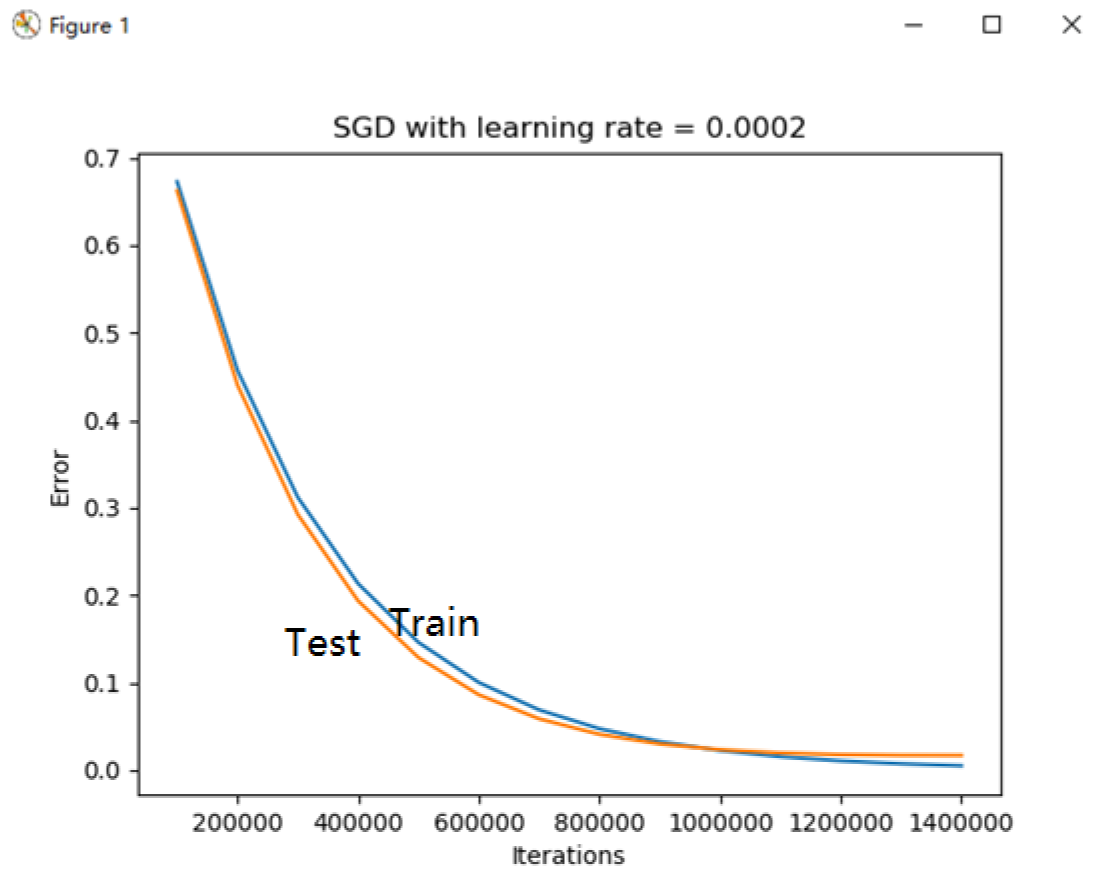
```
def test(a, b, c, data):

    length, n = data.shape
    error = 0.0

    for i in range(0, length):
        x = data[i, 0]
        y = data[i, 1]
        z = data[i, 2]
        f = (a * x) + (b * y) + c
        error += (1/float(length))*np.power((f - z), 2)
    return error

def gradient_descent(a, b, c, data, learning_rate):
    gradient_a = 0.0
    gradient_b = 0.0
    gradient_c = 0.0
    length, n = data.shape

    for i in range(0, length):
        x = data[i, 0]
        y = data[i, 1]
        z = data[i, 2]
        gradient_a += (1/float(length)) * (((a * x) + (b * y) + c) - z) * x
        gradient_b += (1/float(length)) * (((a * x) + (b * y) + c) - z) * y
        gradient_c += (1/float(length)) * (((a * x) + (b * y) + c) - z)
    next_a = a - (learning_rate * gradient_a)
    next_b = b - (learning_rate * gradient_b)
    next_c = c - (learning_rate * gradient_c)
    return [next_a, next_b, next_c]
def run():
    trainingData = np.genfromtxt('dataForTraining.txt',
                     delimiter=' ')
    testData = np.genfromtxt('dataForTesting.txt',
                           delimiter = ' ')
    learning_rate = 0.0002

    trainingData = standardize(trainingData)
    testData = standardize(testData)
    a = 0.0
    b = 0.0
    c = 0.0
    num_iteration = 1500000
    errorForTraining = []
    errorForTesting = []
    for i in range(0, num_iteration):
        a, b, c = gradient_descent(a, b, c, trainingData,
learning_rate)
        if i % 100000 == 0 and i != 0:
            errorForTraining.append(test(a, b, c, trainingData))
            errorForTesting.append(test(a, b, c, testData))
    X = []
    for i in range(1, 15):
        X.append(i*100000)
    plt.figure(1)
    plt.plot(X, errorForTraining)
    plt.plot(X, errorForTesting)
    plt.title('BGD with learning rate = 0.0002')
    plt.xlabel('Iterations')
    plt.ylabel('Error')
    plt.show()
if __name__ == '__main__':
    run()
```

【Exercise 3】

本题采用随机梯度下降法，学习率为 0.00015。与之前两题不同的是，随机梯度不用遍历训练集中的每一个样本，每次迭代只需要随机选一个进行更新参数即可。效果如图：由于随机挑选数据训练，导致收敛的速度比之前慢很多，但是计算时间也要短很多，最后也达到了收敛，并且两个数据集的测试结果差不多，这种方法对于数据量很大的训练集很有用。



当增大学习率，训练误差的收敛速度也更快了。



代码：

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import *


# z = ax+by+c

def standardize(X):
    m, n = X.shape
    for i in range(0, n):
        arr = X[:, i]
        meanVal = arr.mean(axis=0)
        std = arr.std(axis=0)
        if std != 0:
            X[:, i] = (arr-meanVal)/std
        else:
            X[:, i] = 0
    return X

def test(a, b, c, data):

    length, n = data.shape
    error = 0.0

    for i in range(0, length):
        x = data[i, 0]
        y = data[i, 1]
        z = data[i, 2]
        f = (a * x) + (b * y) + c
        error += (1/float(length))*np.power((f - z), 2)
    return error

def gradient_descent(a, b, c, data, learning_rate):
    gradient_a = 0.0
    gradient_b = 0.0
    gradient_c = 0.0
    length, n = data.shape

    i = random.randint(0, length)
    x = data[i, 0]
    y = data[i, 1]
    z = data[i, 2]
    gradient_a = (1/float(length)) * (((a * x) + (b * y) + c) - z) *
x
    gradient_b = (1/float(length)) * (((a * x) + (b * y) + c) - z) *
y
    gradient_c = (1/float(length)) * (((a * x) + (b * y) + c) - z)
    next_a = a - (learning_rate * gradient_a)
    next_b = b - (learning_rate * gradient_b)
    next_c = c - (learning_rate * gradient_c)
    return [next_a, next_b, next_c]

def run():
    trainingData = np.genfromtxt('dataForTraining.txt',
                      delimiter=' ')
    testData = np.genfromtxt('dataForTesting.txt',
                            delimiter = ' ')
    learning_rate = 0.00015

    trainingData = standardize(trainingData)
    testData = standardize(testData)
    a = 0.0
    b = 0.0
    c = 0.0
    num_iteration = 1500000
    errorForTraining = []
    errorForTesting = []
    for i in range(0, num_iteration):
        a, b, c = gradient_descent(a, b, c, trainingData,
learning_rate)
        if i % 100000 == 0 and i != 0:
            errorForTraining.append(test(a, b, c, trainingData))
            errorForTesting.append(test(a, b, c, testData))
    X = []
    for i in range(1, 15):
        X.append(i*100000)
    plt.figure(1)
    plt.plot(X, errorForTraining)
    plt.plot(X, errorForTesting)
    plt.title('BGD with learning rate = 0.00015')
    plt.xlabel('Iterations')
    plt.ylabel('Error')
    plt.show()
if __name__ == '__main__':
    run()
```